

CGI – Common Gateway Interface

"Web Application"

e.g. yahoo email, google, gmail, amazon

Partly, it's the old client/server structure – interface on the client, computation/storage centralized on the server.

Application expressed through a web interface

The most common form of “thin client” app – uses web browser, HTML, HTTP, ... to build the application.

Web Platform

The web is a “platform” choice – it’s an attractive platform, since it works with 95% of computers

Microsoft has 90% marketshare, but that is divided among many OSES – Win95, WinNT, 2000, XP ... – the “web platform” is an attractive way to provide a service that works for all those combinations (not to mention Macs, Linux, Solaris, HPux, ...).

Standards based platform – HTTP, HTML – not tied to any one vendor

Microsoft vs. Web Platform

At some point around 1995, Microsoft became very afraid of the growing “web platform”

If “applications” are coded to the standard web platform, then Win32 has no special advantage vs. other operating systems.

Note the pattern: standards lead to competition – they allow one vendor to be replaced with another. Put another way: standards avoid “vendor lock-in”

Therefore Microsoft wanted to control the browser market. To this end, Microsoft bundled IE with the OS, ... this led to their conviction of illegal monopoly behavior. It is not legal in the U.S. to use monopoly strength in one market (OSes) to take over another market (browsers). Under the Bush administration, the punishment for this conviction has been extremely mild, and the browser continues to be bundled. However, Microsoft’s behavior is no doubt modified by the fear of possible future prosecutions.

In retrospect, the web “application” does not threaten the traditional Win32 application market immediately – that fear was perhaps over-extrapolation of the early growth of the Internet. Someday it will likely threaten Win32 however.

Server 1: Static Pages

HTTP server sits there (as in Homework #2)

Request comes in

Typically HTTP server maps that request into its file system

The server decides what the MIME type is – probably based on its file extension.

This “file system” model is sometimes called “static pages”

Serving static pages is easy – bottleneck is inevitably the network, not the CPU or file system

Server 2: Dynamic Pages

Suppose the server is has a database of books

When a request comes in like “/the+perfect+storm”, or “/wilde”

1. Do some sort of lookup using the words from the request
2. Dynamically generate a page containing the data that matched

How the server interprets the request is up to the server

Point: server can do a computation with a centralized resource, and then send results back to the client in the form of HTML. The interesting part of the computation runs on the server.

.shtml

Some server support this – a low-budget form of server-side processing.

Directives embedded in HTML comments

Processed by server at *request time* – slows things down a bit

e.g. insert mod time of file...

```
<!--#flastmod virtual="foo.html" -->
```

CGI

Common Gateway Interface – a standard which interfaces the HTTP server software with CGI programs which run on the server.

The CGI standard defines how the server communicates all the various facts about the requesting URL (date, referer, accept) to the CGI program, and how the CGI program’s output should be handled and sent back to the requesting client.

CGI is language independent: C, Perl, Java, Unix shell script.

Security

Security concerns – because the CGIs run on the server, they need to take care not to allow a rogue client to compromise the server either by hurting the server or serving up information which was supposed to be secret.

CGI Steps

1. URL

The client has a URL as usual. However , the URL identifies not a page of HTML, but a CGI program on the server.

e.g. `http://cgi.stanford.edu/~ronyeh/cgi-bin/hello.pl`

2. Client GET

The client does a GET request as usual

`GET /cgi-bin/nick/hello.pl HTTP/1.0`

Note that the client is not necessarily doing anything special for this request vs. a static page type request.

3. Server → CGI

The server looks at the GET request, realizes it's a CGI.

4. Run CGI

Server runs the CGI program, feeding it the GET request as input. The GET information is fed to the CGI through environment variables.

Alternately, if the client does a POST, then the data is fed to the CGI via the CGI program's standard input.

5. CGI program

The CGI looks at the input and computes what it wants to compute.

6. HTTP Header

First, the CGI prints the HTTP header fields it wants, followed by a blank line.

7. HTML content

Then the CGI prints the body content it wants and then exits

8. HTTP Server

The HTTP server gathers the output of the CGI, adds more fields to the header, and sends it along to the client.

Trivial Example — hello.pl

```
#!/usr/bin/perl -w

## Hello.pl -- demonstrate a trivial CGI that prints
## out some HTML and the current time on this server.

use strict 'vars';

my($EOL) = "\015\012";

## This is a human-readable str of the current time
my($nowStr);
$nowStr = localtime();

## This line must be included in the header
print "Content-type: text/html$EOL$EOL";

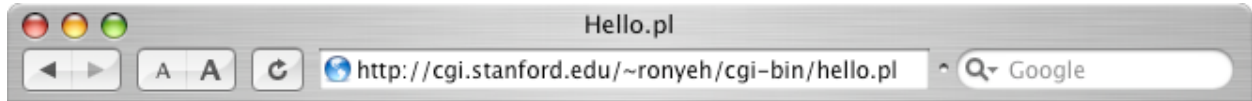
## Write out the HTML content
print "<html><head><title>Hello.pl</title></head>\n";
print "<body bgcolor=white>\n";
print "<h1>Hello.pl</h1>\n";
print "Hello there from CGI-land. It's currently '$nowStr'\n";

print "</body></html>\n";
```

hello.pl CGI Process

This CGI doesn't look at any input, it just produces an HTTP header followed by the same HTML body every time.

For its HTTP header, it just writes the "content-type: text/html" line followed by a blank line. The HTTP server fixes up the rest of the header on its way out to the client.



Hello.pl

Hello there from CGI-land. It's currently 'Wed Apr 28 01:48:12 2004'

Input to CGI programs

Before the CGI is run, its environment variables are set to communicate the request — the primitive Unix method for programs to communicate with each other.

Most likely, the CGI runs as a user called “WWW” or “CGI” which has limited read/write permission on the system. This helps limit some of the security danger, even if the CGI is poorly written.

GET Method

With the “GET” method, the client does a regular GET request to the server, and the data for the CGI is after the “?” in the request URL

The QUERY_STRING environment variable will be the text after the ? in the URL. There are many other environment variables set to indicate other things about the request – see the DumpEnv example below.

POST Method

The client sends the data on the socket after the blank line

The web server sends the data to the CGI on the CGI's standard input

<<EOT; Perl Syntax

The following syntax puts the raw text into the \$string.

The <<EOT; marks the start – the data begins on the next line.

Instead of “EOT” it could be any symbol the programmer likes, however I use EOT since that’s the old programmer way of saying “end of text”.

The “EOT” to end the text must appear on a line by itself with no extraneous whitespace.

```
$string = <<EOT;
```

```
This here can be any old thing.
```

```
Including $variable interpolation.
```

```
*** yeee haaa ***
```

```
EOT
```

```
## The ending EOT must appear on a line by itself with no extra whitespace
```

DumpEnv Perl Script

```
#!/usr/bin/perl
# Print out the values of all the environment variables
# in an HTML <ul>.
# Call from the shell or invoke as a CGI script.

# HTTP header section
print "content-type: text/html\r\n\r\n";

$header = <<EOT;
<html>
<head><title>DumpEnv</title></head>
<body bgcolor=white>
EOT

$trailer = <<EOT;
</body>
</html>
EOT

# Emit an HTML <ul> for all the environment vars
# set up for the CGI
print $header

print "<ul>\n";
## iterate over the keys, but sort them first
foreach $key (sort (keys %ENV)) {
    print "<li><b>$key</b> = $ENV{$key}\n";
}
print "</ul>\n";
print $trailer;
```

Environment Vars

Query_String (after the ? in the URL)

Script_Name (eg /cgi-bin/sbox/~ronyeh/cgi-bin/dumpenv.pl)

Request_Method (usually GET)

Path_Info (extra path after the name of the CGI in the URL)

Remote_Host – The client's IP or DNS address – from a privacy point of view, this is one thing the server will definitely know – how else will it send the packets back to you but by knowing your IP addr?

DumpEnv Output

http://cgi.stanford.edu/~ronyeh/cgi-bin/dumpenv.pl/extrapathinfo?query=hi&pi=3.14159...#fragment



HTML Forms

Present choices to the user in web page

action=url – specifies the URL of the CGI that gets the data

<input name="a-name" ... > -- maps to a form element on screen

Submit – send the data in the form to the CGI – name1=value1&name2=value2... encoded. Uses URL %xx encoding for funny chars, uses = to map names to values, and & syntax between pairs

Trivial Form Example

```
<html>
```

```
<head>
```

```
<title>Form1</title>
```

```
</head>
```

```
<body bgcolor=white>
```

```
<!-- form tag -->

<h1>Form1</h1>

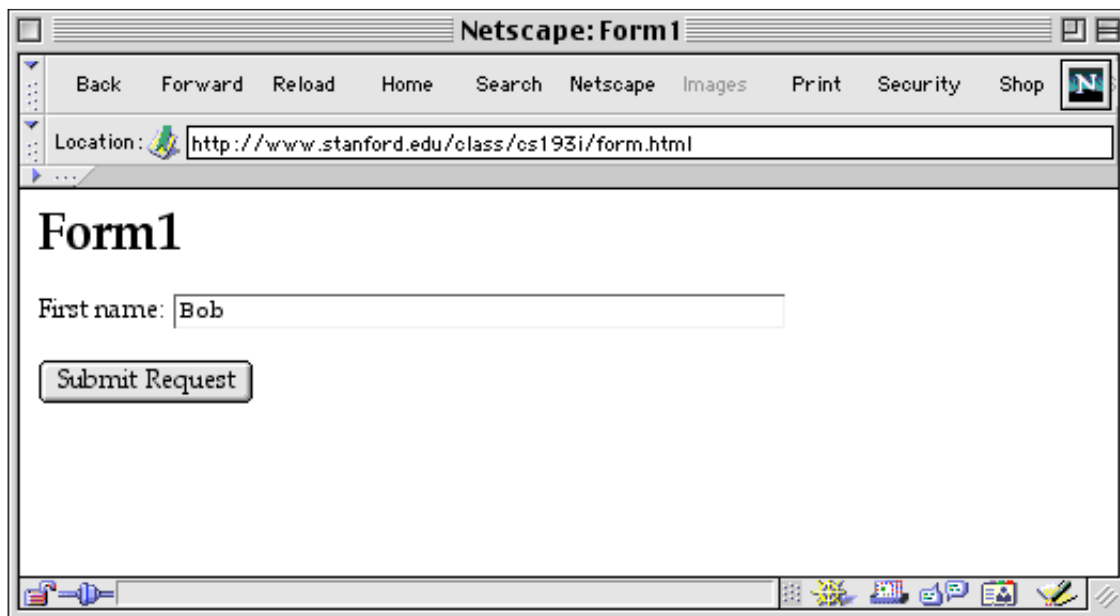
<form
  action=http://www-cs-faculty.stanford.edu/cgi-bin/nick/dumpenv.pl
  method=get>

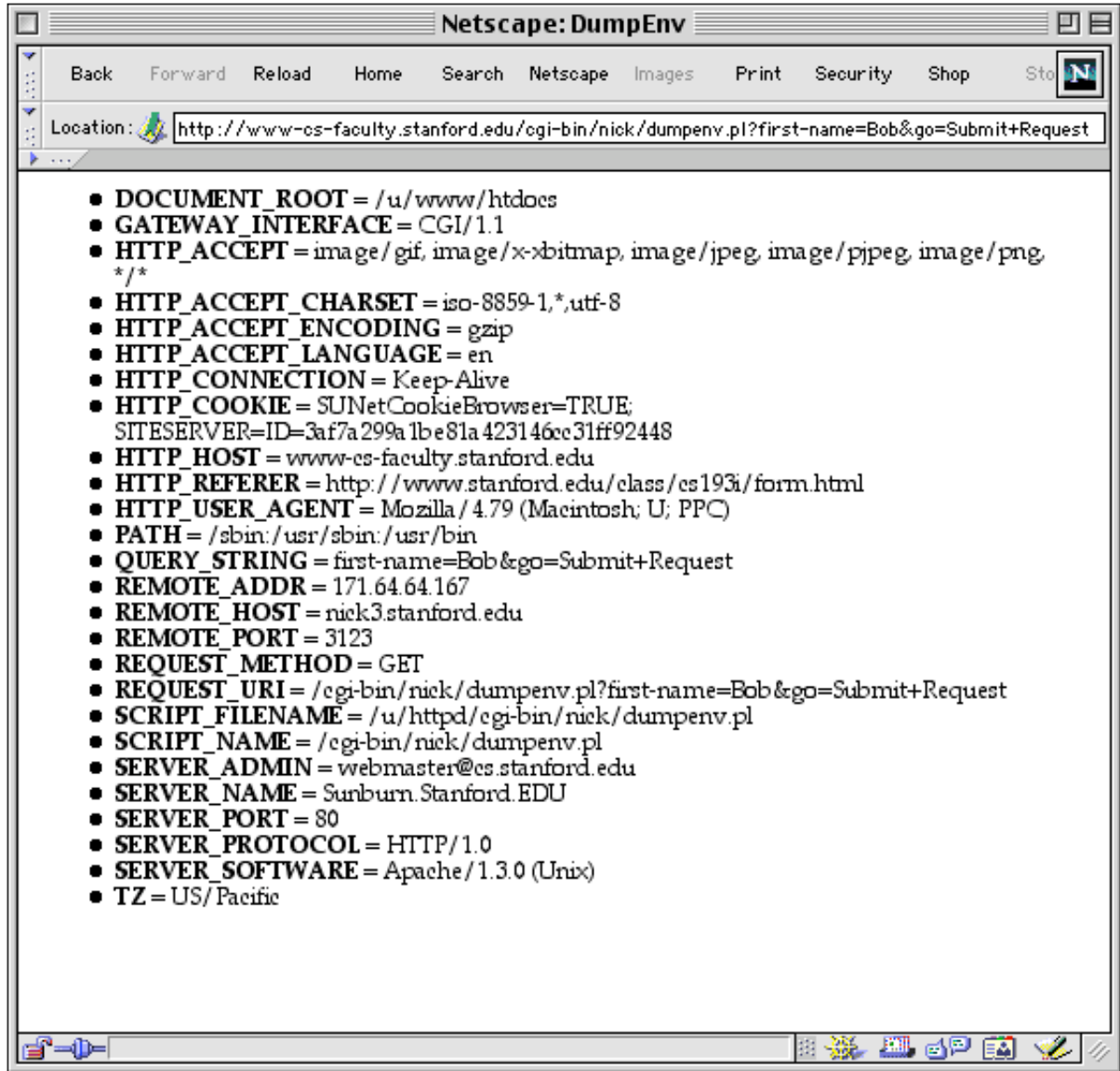
<!-- text input -->
<p>
First name:
<input type=text name = "first-name" size = 40 value="Bob">

<!-- submit button -->
<p>
<input type=submit name=go value="Submit Request">

</form>

</body>
</html>
```





Form – Fields, Submit, Bindings

HTML form – presents GUI text fields, lists, checkboxes,

The user can enter data and "submit" the form.

Submit takes up the state the user has put into the various fields, and sends it to a CGI on the server. The data is encoded as a series of name=value "bindings"

Bindings

A name=value syntax used to encode what the user has entered in the form.

Syntax: `var1=value1&var2=value2&var3=value3`

CGI Environment

Recall Environment variables – combination of client and server info for the CGI to use.

`QUERY_STRING` holds the bindings with the GET method

Forms

An HTML form accepts user input (text fields, checkboxes...) on the client side, and sends the data to a CGI on the server side when the “submit” button is selected.

```
<form action = cgi-url method = get/post>....</form>
```

On submission, sends data to server script named in the action. If the action is omitted, sends data to the URL which produced the form itself.

Form Data Encoding

The data is coded as attribute/value pairs (bindings) with other characters “URL encoded”: space changed to a '+', bindings separated by '&', and non alpha-numeric characters encoded with the "%xx" form where xx is the hex code for the character.

```
person=Chuck+Jones&age=35&topping=anchovies
```

Each form element has a name which is used to identify it in the bindings such as "person" and "age" in the above example.

JavaScript can be used to make a form a little more reactive before the submit – we will study JavaScript a little at the end of the quarter.

The MIME type for the above (+, =, &) encoding is application/x-www-form-urlencoded. Some browsers also support an ENCTYPE=multipart/form-data submission where the form is encoded like a MIME mail enclosure and then submitted by the POST method.

GET Method

GET method.

Attribute/value pairs attached to URL after the ?. This is the simple approach - ok for small amounts of data. Has the advantage that *CGI operations can be bookmarked*. Easier to debug, since you can see the data.

```
http://blah.com/cgi-bin/foo.pl?person=Chuck+Jones&age=35
```

POST Method

POST method.

The bindings are sent from the client to the server on the connection socket. The client sends the usual request stuff but with POST instead of GET, followed by a blank line, followed by the encoded data. The server, in turn, sends the data to the CGI on the CGI's standard input (instead of on the QUERY_STRING env var).

dumpform.pl

Returns to you the name=value bindings from a form submission – use for debugging.

Accepts both GET and POST submissions

```
http://www-cs-faculty.stanford.edu/cgi-bin/nick/dumpform.pl?a=b
```

Form Input Fields – Name and Type

Generally, each input field has a **name** and a **type**

The type is the sort of control

The name is how the data will be identified in the bindings.

type=text

```
<p>
<input type=text name=insult size = 40 value="This sucks">
```

name

name=insult defines the name used to return the data binding, not shown in client HI.

mapping

Maps to insult=*whatever-they-type-in-the-field* when sent to server.

size

Width in characters of the input box

maxLength

max length in characters of the entered string

value="This sucks"

to sets default text

type=password

similar but doesn't show characters on screen

type=checkbox

On-or-off

```
<input type=checkbox name=rude checked>Rude Mode
```

mapping

Maps to rude=on if checkbox is checked, otherwise there is no binding. There's no automatic title around the checkbox, so you need to include your own title as HTML near the checkbox.

value=

If there's a value binding, that is used instead of the string "on"

type=radio

One-of

All have the same name and different values.

One can be checked

```
<p>
```

```
Insult Size:
```

```
<input type=radio name=size value = "small">Small
```

```
<input type=radio name=size value = "medium" checked>Medium
```

```
<input type=radio name=size value = "large">Large
```

mapping

Maps to size=medium

<select> Pop-Up

```
Color:<br>
```

```
<select name = "color">
```

```
<option>red
```

```
<option selected>blue
```

```
<option>green
```

```
<option>purple
```

```
<option>pink
```

```
</select>
```

<option>

Contains option tags followed by text

size=10

How large should the list UI appear (different from the number of items)

multiple

<select multiple ...> allows multiple selections which map to animal=Stoat&animal=Goat
value=

Can have value= within an option, otherwise uses text as value

"selected" pre-selects one

Mapping

Maps to color=blue

<textarea>

Big text area for typing...

```
<textarea name = "comments" rows = rows cols = columns> ...</textarea>
```

Example...

```
<br><textarea name=comments rows=4 cols=60 wrap>
```

type=submit

Submit the data to the action url. If it is given a name (optional), the clicked button is given a binding in the submission — handy if you have multiple buttons and you need to know which was clicked.

```
<input type=submit name=sub value="Submit Insult Request">
```

Trick: if there's a single text field in the form, hitting return can also do a submit – this feature varies among browsers.

Can have multiple submit buttons, all with the same name, and distinguish which got clicked by the value. Or, could give each submit button a unique name.

type=image

Variant on the submit button – add a src= def in the tag

The image may show up with some sort of visual cue in the browser that it is clickable.

Can add width=xxx or height=yyy to scale the image (as with a tag

Maps to name.x=27 name.y=67 in the bindings (it would be nice to also get a sort of name=1 binding, but that's not part of the spec)

```
<input type=image name= jeffbutton src=jeffsad.jpg>
```

type=hidden

```
<input type = "hidden" name = "secret" value ="blorg">
```

Maps to secret=blorg in data, but doesn't show in client HI — a good way to store information in the form for later reading by your CGI.

type=reset

Clear all fields in client HI

I have never in my life gotten any use out of this button.

```
<input type = "reset" value = "button-title">
```

HTML Form Example

```

<html>

<head>
<title>Sample Form</title>
</head>

<body bgcolor=white>

<!-- form tag -->
<!-- action=url to send data, method=get or post -->
<form
  action=http://www-cs-faculty.stanford.edu/cgi-bin/nick/dumpform.pl
  method=get>

<h1>Insult Generator</h1>

<!-- text input -->
<!-- maps to person="Bob" -->
<p>
First name:
<input type=text name=person size = 40 value="Bob">

<!-- checkbox -->
<!-- maps to size=on if checked (not present if not checked) -->
<p>
<input type=checkbox name=rude>Rude mode

<!-- radio -->
<!-- maps to size=medium -->
<p>
Insult Size:
<br><input type=radio name=size value = "small">Small
<br><input type=radio name=size value = "medium" checked>Medium
<br><input type=radio name=size value = "large">Large

<!-- selection pop-up list -->
<!-- maps to animal=Goat -->
<p>
Preferred Animal:
<select name = "animal">
<option>Stoat
<option selected>Goat
<option>Weasel
</select>

<!-- selection scrollable list -->
<!-- maps to color=pink -->
<p>
Color:<br>
<select name = "color" size=8>
<option>red

```

```
<option selected>blue
<option>green
<option>purple
<option>pink
<option>yellow
<option>striped
<option>brown
<option>mauve
<option>dotted
<option>speckled
<option>puce
<option>orange
<option>tan
<option>black
<option>white
<option>gray
</select>

<!-- hidden field -->
<input type=hidden name=secretcode value="Keyser Soze">

<!-- text area -->
<p>
Comments:
<br><textarea name=comments rows=4 cols=60 wrap>
</textarea>

<!-- submit button -->
<!-- maps to sub="Submit Insult Request" if clicked -->
<p>
<input type=submit name=sub value="Submit Insult Request">

<!-- image submit -->
<!-- maps to im-submit.x=xxx for click location if clicked -->
<p>
<input type=image name=im-submit src=jeffsad.jpg width=100>

</form>

</body>
</html>
```

Sample Form
http://www.stanford.edu/class/cs193i/insult/ Google

Insult Generator

First name: ASuckyStudent

Rude mode

Insult Size:

- Small
- Medium
- Large

Preferred Animal: Weasel

Color:

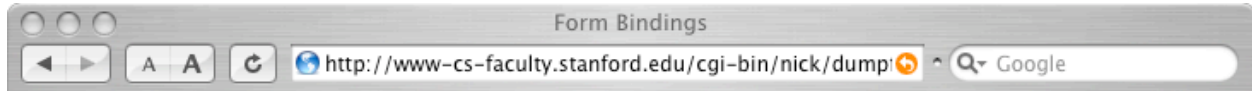
- blue
- green
- purple
- pink
- yellow
- striped
- brown
- mauve

Comments:

Hahahah you suck. A lot.

Submit Insult Request





Your Key/Value Bindings...

animal	Weasel
color	striped
comments	Hahahah you suck. A lot.
person	ASuckyStudent
rude	on
secretcode	Keyser Soze
size	large
sub	Submit Insult Request

CGI.pm

Very widely used standard form processing module for Perl.

use CGI;

Modern Perl installations have CGI.pm installed already by default

It has lots of features, but for now we'll just use it to extract bindings.

use CGI;

my \$q = new CGI;

\$param = \$q->param('param-name');

Will be undef if there is no such binding.

Use the defined() operator to check it

If there are multiple bindings for a variable...

\$q->param("var") returns an array

Assigning into a scalar just takes the first from the array

Assign into an array to see all the values

\$q->param in an array context returns an array of all the form variable names, so you can iterate over them all.

dumpform.pl code

```
#!/usr/bin/perl -w
# Extract all the bindings in a form submission, and put
# them into a little table.

use CGI;

my $q = new CGI;

print $q->header("text/html");

# Print out all the key/value pairs....

print "<html><head><title>Form Bindings</title></head><body bgcolor=white>\n";
print "<h1>Your Key/Value Bindings...</h1>\n";

print '<table border=1 width="100%">'; ## note use of ' to hide " in string

my(@vars, $var, $val);
@vars = $q->param;

foreach $var (sort @vars) {
    $val = $q->param($var);
    print "<tr>\n"; ## one <tr> for each row
    print "<td><b>$var</b></td>\n"; ## one <td> for each elem in row
    print "<td>$val</td>\n";
    print "</tr>\n";
}

print "</table>\n";

print "</body></html>\n";
```